Data oddania sprawozdania: 13.06.2018

Jacek Woźniak 234995 Jakub Majewski 238902 Termin zajęć: środa TP 7:30 Prowadzący: Mgr Inż. Dominik Żelazny

SPRAWOZDANIE Organizacja i Architektura Komputerów Projekt

I. Cele projektu

Napisać program realizujący zadanie interpretera języka BF.

II. Założenia

- 1. Program został napisany w całości w języku assembly.
- 2. W programie znajduje się opcja wczytywania pliku źródłowego z kodem BF.
- 3. W programie znajduje się opcja wczytywania kodu BF z konsoli.
- 4. Program zawiera w sobie menu umożliwiające wybranie źródła kodu przeznaczonego do interpretacji.

III. Język BF

1. Geneza

Język BF (skrót od Brainf*ck) jest językiem ezoterycznym programowania, który został stworzony około 1993 roku przez Urbana Müllera. Składnia języka jest bardzo uboga co wyróżnia go na tle innych. Interpretacja opiera się na prostym modelu maszyny taśmowej z przesuwnym wskaźnikiem (idea podobna do maszyn Turinga).

2.Instrukcje języka BF ze swoimi odpowiednikami w języku C

Znak	Znaczenie	Odpowiednik w C
>	inkrementuje wskaźnik	++p
<	dekrementuje wskaźniki	p
+	inkrementuje wartość w komórce wskazywanej przez wskaźnik	++(*p)
-	dekrementuje wartość w komórce wskazywanej przez wskaźnik	(*p)
•	wyświetla znak w ASCII o wartości wskazywanej komórki	putchar(*p)
,	pobiera znak i wstawia go do wskazywanej komórki	*p=getchar()
[początek pętli, która nie wykona się jeżeli w aktualnie wskazanej komórce znajduje się wartość 0	while(*p){
]	koniec pętli	}

3. Przykładowy program typu "Hello world!" napisany w języku BF z odpowiednikami w języku C oraz aktualnym stanem taśmy (* oznacza aktualnie wskazywaną komórkę):

```
[*0][0][0][0]
++++++++
                      *p+=10;
                                                                [*10][0][0][0][0]
                      while(*p) {
                         ++p; *p+=7;
                                                                ---
>++++++
                         ++p; *p+=10;
                                                                ---
>++++++++
                         ++p; *p+=3;
                                                                ---
>+++
                         ++p; *p+=1;
>+
                         p-=4; --*p;
<<<<-
                                                                _ _ _
                      }
                                                                po pętli (w ascci):
]
                                                                [*0]['F'] ['d'] [' '-2]['\n']
[0] [*'H']['d'] [' '-2]['\n']
[0] ['H'] [*'e'][' '-2]['\n']
>++.
                      ++p;
                              *p+=2;
                                       putchar(*p); // 'H'
                                       putchar(*p); // 'e'
                      ++p;
                              *p+=1;
                                                                [0] ['H'] ['1'] [' '-2]['\n']
                                      putchar(*p); // 'l'
                              *p+=7;
++++++.
                                                                [0] ['H'] ['1'] [' '-2]['\n']
                                       putchar(*p); // 'l'
                                                                [0] ['H'] ['o'] [' '-2]['\n']
                                      putchar(*p); // 'o'
                              *p+=3;
                                      putchar(*p); // ' '
                                                                [0] ['H'] ['o'] [*' '] ['\n']
>++.
                      ++p;
                              *p+=2;
                                                                [0] [*'W<sup>-</sup>]['o'] [' '] ['\n']
                              *p+=15; putchar(*p); // 'W'
p-=2;
                                                                [0] ['W'] [*'o'][' ']
                                                                                        ['\n']
                                       putchar(*p); // 'o'
>.
                      ++p;
                                                                [0] ['W'] [*'r'][' ']
                                       putchar(*p); // 'r'
                                                                                        ['\n']
['\n']
                              *p+=3;
+++.
                                                                [0] [,M,] [*,1,][, ,]
                                       putchar(*p); // 'l'
                              *p-=6;
                                                                [0] ['W'] [*'d'][' '] ['\n']
                                       putchar(*p); // 'd'
                              *p-=8;
                                      putchar(*p); // '!'
                                                                [0] ['W'] ['d'] [*'!<sup>†</sup>] ['\n']
                              *p+=1;
>+.
                      ++p;
                                       putchar(*p); // '\n'
                                                                [0] ['W'] ['d'] ['!'] [*'\n']
>.
                      ++p;
```

IV. Realizacja

Etap 1.

- Napisać działający interpreter języka BF w języku assembly, czyli rdzeń (core) dla całego przyszłego programu.
- Uruchomić kilka gotowych programów w celu sprawdzenia działania interpretera.
- Jedyny sposób wprowadzenia kodu do interpretacji to zmiana kodu źródłowego i ponowna kompilacja.

Etap 2.

Dodać możliwość wczytywania kodu programu BF z pliku.

Do wczytywania została wykorzystana kombinacja funkcji systemowych:

SYS_OPEN, SYS_READ, SYS_CLOSE

Kluczowy fragment kodu odpowiedzialnego za wczytywanie danych z pliku:

```
movq 8(%rsi), %rdi # path
mov1 $0, %esi  # flags
mov1 $0666, %edx  # mode
mov1 $2 %eav
mov1 $2, %eax # SYS_OPEN
syscall
movl %eax, %ebx
                     # save fd
# read the file
movl %eax, %edi  # fd
leaq buf, %rsi  # buf
movl $bufSize, %edx # count
movl $0, %eax # SYS_READ
syscall
# close the file
movl %ebx, %esi  # fd
movl $3, %eax  # SYS_CLOSE
syscall
exit:
xor %eax, %eax
                      # return 0
pop %rbx
                      # restore rbx
```

Etap 3.

Dodać możliwość wczytywania kodu programu BF z konsoli, oraz proste menu umożliwiające wybranie źródła do interpretacji (z pliku lub z konsoli). Do wczytywania z konsoli ciągu znaków przeznaczonego do interpretacji została wykorzystana funkcja systemowa SYS_READ.

Kluczowy fragment kodu odpowiedzialnego za wczytywanie ciągu znaków z konsoli:

```
#pobranie tekstu do dalszej
obróbki
movq $SYSREAD, %rax
movq $STDIN, %rdi
movq $buf, %rsi
movq $bufSize, %rdx
syscall
```

V. Wnioski

Realizacja projektu pozwoliła na lepsze zrozumienie języka assembly oraz tego jak działa komputer od środka. Ponad to umożliwiła poszerzenie wiedzy o pracy samego systemu operacyjnego.